Partikelschwarmoptimierung am Beispiel des Traveling Salesman Problem, SS16

Seminararbeit, Semester 4

als Prüfungsleistung im Rahmen des Seminars

Bearbeitet von:
Björn Bäcker
Daniel Fischer
Thomas Gross
Christoph Schmid

Betreuer:
Prof. Dr. habil. Thomas Thierauf

Hochschule Aalen
Fakultät Elektronik und Informatik
Studiengang Informatik

Inhaltsverzeichnis

Abbildungsverzeichnis

Eidesstattliche Erklärung

Hiermit versichern wir, Björn Bäcker, Daniel Fischer, Thomas Gross und Christoph Schmid, dass die vorliegende Seminararbeit, gemäß § 35 Abs. 1 SPO 29 der Hochschule Aalen, selbstständig und unter ausschließlicher Verwendung der angegebenen Quellen und Hilfsmittel erstellt wurde.
Die Arbeit wurde bisher keiner anderen Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt und auch nicht veröffentlicht.

22. Juni 2016

Björn Bäcker

Daniel Fischer

Christoph Schmid

Thomas Gross

Zusammenfassung

Im Rahmen dieser Arbeit wird die Partikelschwarmoptimierung (PSO) vorgestellt und anschließend auf das Traveling Salesman Problem (TSP) angewandt. Im Anschluss werden das Brute-Force-Verfahren, das Nearest-Neighbor-Verfahren und der Ameisenalgorithmus zur Lösung des TSP kurz vorgestellt und gegen Ende mit der Lösung per PSO gegenübergestellt.

Abstract

In the context of this thesis, the Particle Swarm Optimziation gets introduced. After that, it will be used to solve the Traveling Salesman Problem, followed by the introduction of the Brute-Force-Procedure, the Nearest-Neighbor-Procedure and the Ant Colony Algorithm, which all can be used for the solution of the Traveling Salesman Problem. Finally, these procedures will be compared to the Particle Swarm Optimization, to determine the most fitting algorithm for this problem.

1 Einführung

Optimierungsprobleme stellen einen Bereich der Informatik dar, der schon von Beginn an von großer Bedeutung war. Im Lauf der Zeit wurden unterschiedlichste Verfahren entwickelt, um diese so effizient wie möglich zu lösen. Eines der hieraus entstandenen Heuristiken für die Lösung von Optimierungsproblemen ist die Partikelschwarmoptimierung.
Diese wurde 1995 von Kennedy und Eberhart für kontinuierliche Probleme entwickelt, ist aber nicht auf diese beschränkt und wurde bereits von einigen anderen Wissenschaftlern für andere Probleme optimiert [Sie15].
Eines dieser Probleme ist das Traveling Salesman Problem. Es gehört zu den ältesten und populärsten Problemen der Informatik und doch hat es bis heute nicht an Relevanz verloren, da es aufgrund seines wirtschaftlichen Nutzens in den Bereichen der Logistik, der Touristik, der Produktion von Mikrochips et cetera nach wie vor eine bedeutende Rolle spielt [Sch99].

Im Rahmen dieser Arbeit möchten wir die Partikelschwarmoptimierung vorstellen und evaluieren, inwiefern es im Gegensatz zu alternativen Verfahren für das Traveling Salesman Problem von Nutzen ist.

2 Grundlagen

Um die Funktionsweise der Partikelschwarmoptimierung zu verstehen, betrachten wir vorerst einige Grundlagen zur Optimierung, Evolution, Intelligenz und Emergenz.

2.1 Optimierung

In der angewandten Mathematik steht man oft vor dem Problem, eine optimale Kombination von Parametern bestimmen zu wollen. Bei Operationen mit vielen Parametern und vielen Einstellungen pro Parameter kann dies sehr zeitaufwendig werden.
Schaut man sich als Beispiel eine Produktionsmaschine an, die 10 Parameter mit jeweils 100 Einstellungen besitzt, müsste man 10^{100} Möglichkeiten durchrechnen.
Dieses Problem kann mittels Erfahrung und Statistik vereinfacht werden, indem man eine Funktion $f : Parameter \rightarrow Güte$ [Mel08] aufstellt. Dieses Verfahren ist ein Teilgebiet der angewandten Mathematik und heißt *Operations Research*.
Dieses klassische Optimierungsverfahren stößt jedoch an seine Grenzen, wenn die untersuchten Funktionen multimodal oder nicht differenzierbar sind. Für diese Funktionen müssen neue Verfahren und Algorithmen gefunden werden. Schaut man sich die Natur an, kann man überall Lebewesen entdecken, die sich an die unwirklichsten Lebensräume angepasst haben. Daraus folgt, dass die Natur sehr gut darin ist, sich selbst zu optimieren. Die Optimierungsstrategie der Natur wird als Evolution bezeichnet und ist hoch komplex. Evolution kann aber in die Vorgänge *Rekombination*, *Mutation* und *Selektion* aufgeteilt werden [Mel08].

2.2 Evolution

Die biologische Evolution basiert auf Rekombination, Mutation und Selektion. Die Rekombination ermöglicht neue Kombinationen von erfolgreichen Merkmalen des Elternpaares. Erfolgreiche Kombinationen werden von der Natur daran erkannt, dass die Eltern das Fortpflanzungsalter erreicht haben. Daraus ergeben sich für die neue Generation bessere Überlebenschancen. Mutation ist ein spontaner Prozess, bei dem sich Merkmale zufällig minimal verändern. Dies wird in den meisten Fällen von dem Organismus repariert. In den wenigen Fällen, in dem die Mutation nicht entfernt wird, können neue Spezies entstehen. Da Mutationen auf Zufall basiert, können diese sowohl positive als auch negative Effekte auf den Organismus haben. Diese Effekte

werden durch Selektion aussortiert, da nach Darwin sich nur die am besten angepassten Spezies durchsetzen. Das Verfahren der Selektion wird auch *Survival of the Fittest* genannt [Mel08].
Aus der natürlichen Evolution können evolutionäre Algorithmen abgeleitet werden, welche diese drei Vorgänge umsetzen. Diese Algorithmen sind sogenannte *Meta-Heuristiken* und können zur Optimierung eingesetzt werden. Sie sind meist nur hinreichend genau und finden nicht zwingend das globale Minimum, aber man ist damit in der Lage, auch exponentielle Probleme hinreichend genau zu optimieren.
Die Evolution wird mit N Individuen simuliert, die zu Beginn zufällig im Lösungsraum verteilt werden. Anhand einer Fitnessfunktion wird die Lösung am Ende jeder Iteration bewertet. Zu Beginn der neuen Iteration werden meist 50% der Individuen gewählt, die die neuen Eltern der neuen Generation werden. Ausgewählt werden die Individuen mit den besten Ergebnissen. Nach der Auswahl der Eltern werden diese nun rekombiniert oder geklont. Im Anschluss wird die neue Generation mit einer gewählten Wahrscheinlichkeit der Zielrichtung leicht verändert. Dies stellt die Mutation der natürlichen Evolution dar [Mel08].

2.3 Intelligenz

Die Intelligenz ist die Fähigkeit, Probleme zu lösen und sich an neue Gegebenheiten anzupassen. Allgemein wird das Gehirn als Sitz der Intelligenz angesehen und besteht aus Nervenzellen, die jeweils nur ein Signal verarbeiten können. Daraus folgt, dass Intelligenz die Kommunikation zwischen den Zellen ist.
Das Produkt der Intelligenz sind Gedanken, welche auch der Evolution unterliegen können. Wenn Gedanken weitergegeben werden, können diese rekombiniert und mutiert werden. Je schneller die Kommunikation der Gedanken, desto schneller ist auch die Evolution der Gedanken.
Dieses Prinzip kann auf Schwärme übertragen werden, woraus die Theorie der Sachwarmintelligenz entsteht. Die Intelligenz des Schwarmes ist nicht die Summe der Intelligenzen der Einzelindividuen, sondern vielmehr die Fähigkeit und Qualität eines Schwarms, gemeinsam zu agieren.
Graig Reynol gelang es 1987 anhand von drei Regeln einen Vogelschwarm und somit eine Sachwarmintelligenz zu simulieren. Die drei Regeln lauten:

1. Weiche Artgenossen und Hindernissen aus.
2. Versuche mit der gleichen Geschwindigkeit wie dein Nachbar zu fliegen.
3. Versuche näher an das Zentrum des Schwarmes heranzukommen.

Entdeckt nun ein Individuum zum Beispiel eine Futterquelle, richtet sich das Individuum leicht darauf aus, was wiederum Auswirkungen auf dessen Nachbarn hat, die sich nun auch in diese Richtung ausrichten. Je mehr sich der Schwarm nun in diese Richtung ausrichtet, desto mehr Individuen entdecken die Futterquelle und richten sich verstärkt darauf aus. Daraus folgt, dass sich der ganze Schwarm zu der Futterquelle ausrichtet und diese auch erreicht [Mel08].

2.4 Emergenz

Wenn sich einzelne Elemente so beeinflussen, dass sich eine geordnete Struktur ergibt, so nennt man dies Emergenz.
Das bedeutet, dass ein Kollektiv in der Lage ist, Probleme zu lösen, ohne dass die Definition des Problems oder der Lösungsweg vorliegt. Auch unser Gehirn unterliegt diesem emergenten Effekt, da die Intelligenz aus der Kommunikation der Zellen entsteht, ohne dass Strategien zur Problemlösung vorliegen [Mel08].

3 Partikelschwarmoptimierung

James Kennedy und Russel Eberhart haben aus den vorausgegangenen Beobachtungen der Natur erstmals 1995 die Partikelschwarmoptimierung auf der IEEE Konferenz für neuronale Netzwerke vorgestellt [JK95]. Das Verfahren beruht auf dem Verhalten von Fischen oder Vögeln in Schwärmen, in denen sich die Tiere synchron und kollisionsfrei bewegen. Dieses Verhalten wurde nun auf ein Problem übertragen, in dem sich Partikel (die Tiere) in einem Suchraum bewegen, um ein Problem zu lösen.

3.1 Partikel und Suchraum

Um das Verfahren auf ein mathematisches Problem anwenden zu können, müssen zunächst der Suchraum und die Partikel definiert werden [PV10]. Zur Verdeutlichung wird die Optimierung eines Otto-Motors als Beispiel mitgeführt.

3.1.1 Suchraum

Der Suchraum $A \subset \mathbb{R}^m$ ist ein mehrdimensionaler Raum, in dem sich die Partikel bewegen. Dieser ist beschränkt, das heißt er hat eine endliche Anzahl an möglichen Positionen, die die Partikel besuchen können. Die Funktion $f : A \to \mathbb{R}$ ist die zu optimierende Zielfunktion, die für jeden Punkt in A eine Lösung in $\mathbb{R}$ besitzt.

Beispiel 3.1 *Angenommen, man möchte bei einem Otto-Motor die Leistung optimieren. Bei dieser Optimierung gibt es verschiedene Werte, die man ändern kann, um die Leistung zu beeinflussen. Beispiele hierfür sind die Drehzahl, der Einspritzdruck und das Verhältnis der Luft zum Benzin. Nimmt man diese Variablen erhält man den Suchraum $A \subset \mathbb{R}^3$, wobei die $d_1 = Drehzahl$, $d_2 = Einspritzdruck$, $d_3 = LuftBenzinGemisch$.*
Die Zielfunktion wäre bei dieser Optimierung zum Beispiel durch die Ergebnisse des Motorprüfstands repräsentiert.

3.1.2 Partikel

In dem Suchraum A bewegt sich nun ein Suchschwarm $S = \{x_1, \ldots, x_N\}$ mit N Partikeln. Der Suchschwarm ist die Menge aller Partikel $x_i \in S$ für $i = 1, 2 \ldots, N$. Jedes Partikel x_i hat folgende Eigenschaften im Suchraum A zum Zeitpunkt t der Iteration:

- es hat eine Position $p_i(t) \in A$ und einen Funktionswert $f(p_i(t))$ für diese Funktion

Beispiel 3.2 *Im Beispiel wäre die Position für x_1 zum Beispiel $p_1(1) = (2000|180|0.4)$. Das Partikel hat dann an dieser Position einen Funktionswert $f(p_1(1)) = 90$. Somit hat der Motor bei den eingegebenen Werten ein Drehmoment von $90Nm$.*

- es hat eine beste Position $p_{b,i}(t)$ im Suchraum, an der das Partikel seinen bisher besten Funktionswert hatte.

  Beispiel 3.3 *Angenommen der obige Wert ist der bisher beste Wert des Partikels, dann ist $p_{b,1}(1) = (2000|180|0.4)$.*

- es kennt die beste globale Position $p_g(t)$, also die Position, an welcher der bisher beste Funktionswert aller Partikel liegt. Diesen Wert kann ein beliebiges Partikel im Suchraum entdeckt haben.

  Beispiel 3.4 *Das Partikel x_4 hat an der Position $p_4(1) = (4000|300|0.6)$ einen Funktionswert von $f_4(p_4(1)) = 130$. Da kein anderes Partikel in dieser oder einer der vorigen Iterationen ein besseres Ergebnis erzielt hat, ist diese Position $p_g(1)$.*

- es hat eine Geschwindigkeit $v_i(t)$, mit der sich das Partikel durch den Raum bewegt.

  Beispiel 3.5 *Angenommen das Partikel hat in diesem Schritt eine Geschwindigkeit von $v_1(t) = \begin{pmatrix} 1280 \\ 140 \\ 0.2 \end{pmatrix}$. Diese resultiert aus den vorherigen Werten von x_1. (Die Berechnung für die Geschwindigkeit wird in Kapitel 3.2 genauer erläutert).*

3.2 Berechnung

Mittlerweile existieren mehrere Standards zur Berechnung der Geschwindigkeit eines Partikels [Cle12]. Hier wird die erste Variante von 1995 behandelt. Die Geschwindigkeit und die Position stellen sich aus folgenden Gleichungen zusammen:

$$v_i(t+1) = v_i(t) + c_1 R_1(p_{b,i}(t) - p_i(t)) + c_2 R_2(p_g(t) - p_i(t)) \quad (1)$$

$$p_i(t+1) = p_i(t) + v_i(t+1) \quad (2)$$

Bei der Berechnung der neuen Geschwindigkeit wird aus der Entfernung der alten Position zum globalen Optimum $p_g(t) - p_i(t)$, zum partikeleigenem Optimum $p_{b,i}(t) - p_i(t)$ und aus der alten Geschwindigkeit die neue Geschwindigkeit berechnet. Dabei fallen die Zufallszahlen $R_1, R_2 \in [0, 1]$ ins Gewicht, sowie die Konstanten c_1, c_2, mit denen das Verhalten des Schwarms verändert werden kann (Abbildung 1). Wenn $c_1 > c_2$ ist, dann streben die Partikel in Richtung ihres lokalen Optimum. Ist $c_2 > c_1$, dann konvergieren die Partikel eher zum globalen Optimum [Ham12].

Beispiel 3.6 *Angenommen man möchte, dass der Suchschwarm eher zum globalen Optimum strebt, dann wählt man beispielsweise* $c_1 = 1$ *und* $c_2 = 2$. *Seien die Zufallszahlen in diesem Beispiel* $R_1 = 0.7$ *und* $R_2 = 0.3$. *Nimmt man die Werte von* x_1 *aus Absatz 3.1.2 so erhält man folgende Formeln:*

$$v_1(2) = v_1(1) + 1 * 0.7(p_{b,1}(1) - p_1(1)) + 2 * 0.3(p_g(1) - p_1(1)) \quad (3)$$
$$p_1(2) = p_1(1) + v_1(2) \quad (4)$$

Setzt man die Werte ein:

$$v_1(2) = \begin{pmatrix} 1280 \\ 140 \\ 0.2 \end{pmatrix} + 1 * 0.7(\begin{pmatrix} 2000 \\ 180 \\ 0.4 \end{pmatrix} - \begin{pmatrix} 2000 \\ 180 \\ 0.4 \end{pmatrix}) + 2 * 0.3(\begin{pmatrix} 4000 \\ 300 \\ 0.6 \end{pmatrix} - \begin{pmatrix} 2000 \\ 180 \\ 0.4 \end{pmatrix})$$
$$= \begin{pmatrix} 2420 \\ 212 \\ 0.32 \end{pmatrix}$$
$$p_1(2) = \begin{pmatrix} 2000 \\ 180 \\ 0.4 \end{pmatrix} + \begin{pmatrix} 2420 \\ 212 \\ 0.32 \end{pmatrix} = \begin{pmatrix} 4420 \\ 392 \\ 0.72 \end{pmatrix}$$

Somit hat man nach den Berechnungen eine neue Position für x_1 *mit* $p_2 = (4420|392|0.72)$, *mit der man den neuen Funktionswert* $f(p_1(2))$ *ermitteln kann.*

Nachdem die neue Position des Partikels bestimmt wurde, wird nun berechnet, ob ein neues Optimum erreicht wurde:

$$p_{b,i}(t+1) = \begin{cases} p_i(t+1), & f(p_i(t+i)) \leq f(p_{b,i}(t)) \\ p_{b,i}(t), & \text{sonst} \end{cases} \quad (5)$$

$$p_g(t+1) = \begin{cases} p_{b,i}(t+1), & f(p_{b,i}(t+1)) \leq f(p_g(t)) \\ p_g(t), & \text{sonst} \end{cases} \quad (6)$$

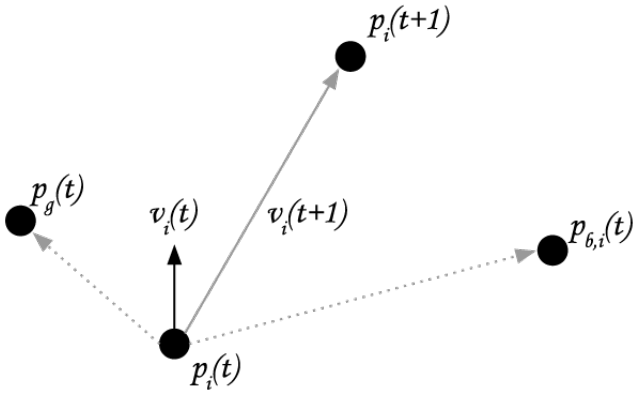

Abbildung 1: Berechnung der neuen Geschwindigkeit und Position eines Partikels

Diese Funktionen optimieren auf ein Minimum hin. Möchte man auf ein Maximum hin optimieren, so ersetzt man die kleiner-gleich-Zeichen durch größer-gleich-Zeichen.

Beispiel 3.7 *Der Funktionswert $f(p_1(2))$ wurde auf dem Prüfstand ermittelt und hat den Wert $f(p_1(2)) = 155Nm$. Da diese Optimierung ein Maximum sucht, müssen die Kleiner-Gleich-Zeichen ersetzt werden. Daraus folgen die Formeln:*

$$p_{b,i}(t+1) = \begin{cases} p_i(t+1), & f(p_i(t+i)) \geq f(p_{b,i}(t)) \\ p_{b,i}(t), & sonst \end{cases}$$

$$p_g(t+1) = \begin{cases} p_{b,i}(t+1), & f(p_{b,i}(t+1)) \geq f(p_g(t)) \\ p_g(t), & sonst \end{cases}$$

Setzt man die Werte ein, so erhält man ein neues lokales und globales Optimum für x_1: $p_{b,1}(2) = p_g(2) = p_1(2)(4420|392|0.72)$

3.3 Algorithmus

Ein einfacher Algorithmus besteht aus 2 Phasen. Die erste Phase initialisiert jedes Partikel im Suchraum an einer zufälligen Position, an der noch kein anderes Partikel ist. Der zweite Schritt ist die Berechnung, bis eine Abbruchbedingung erfüllt ist. Eine Abbruchbedingung kann zum Beispiel ein Optimum sein, das mit einer gewissen Toleranz erreicht wurde, oder dass ein Maximum an Iterationen erreicht worden ist. Aus diesen Überlegungen ergibt sich der Pseudocode:

```
1: p_g := ∞
2: for all x_i ∈ S do
3:     Initialisiere x_i auf Position p_i in A
4:     Setze lokales Optimum auf p_i
5:     Wenn f(p_{b,i}(t)) ≤ f(p_g(t)) setze p_g := p_{b,i}
6:     Setze v_i auf eine zufällige Geschwindigkeit
7: end for
8: while Abbruchbedingung ≠ true do
9:     for all x_i ∈ S do
10:        Berechne neue Geschwindigkeit für x_i
11:        Berechne neue Position für x_i
12:        Berechne neues Optimum p_{b,i} und p_g
13:    end for
14: end while
15: In p_g befindet sich das Optimum
```

In den Zeilen 2 bis 7 wird jedes Partikel auf eine zufällige Position im Suchraum A gesetzt, danach wird das lokale Optimum auf diese Position gesetzt und mit dem vorherigen globalen Optimum verglichen. Ist das lokale Optimum besser als das globale, so wird es als neues globales Optimum gesetzt. Als letzten Schritt der Initialisierung wird die Geschwindigkeit des Partikels auf einen zufälligen Wert gesetzt.
Nachdem alle Partikel initialisiert wurden, beginnt der Algorithmus die Optimierung durchzuführen. Dabei werden in den Zeilen 9 bis 12 für alle Partikel die neue Geschwindigkeit, die neue Position und das neue Optimum nach den Formeln in Absatz 3.2 berechnet. Ist eine der Abbruchbedingungen erfüllt, enthält nun p_g das beste berechnete Optimum der gegebenen Funktion.

3.4 Nachbarschaft

In diesem Algorithmus ergibt sich nun das Problem, dass sich der komplette Schwarm dem Optimum nähert und deswegen seine Diversität verliert, da alle Partikel sich auf dieser Position häufen. Auch kann es passieren, dass sich der Schwarm nur um ein lokales Optimum herum bewegt. Aus diesem Grund wurden die Nachbarschaftsbeziehungen entwickelt.
Bei der Nachbarschaft kennen die einzelnen Partikel ihren Nachbarn und kennen nur aus dieser Gruppe die beste Position. So wird die Diversität erhalten, aber der Schwarm konvergiert langsamer zu einem Optimum.
Es gibt viele Möglichkeiten, die Nachbarschaft zu realisieren und man sollte diese anhand des Problems auswählen.
Diese Möglichkeiten werden im Rahmen dieser Arbeit nicht behandelt [PV10].

4 Traveling Salesman Problem

Im Rahmen dieser Arbeit wird die Partikelschwarmoptimierung verwendet, um das Traveling Salesman Problem zu lösen. Allerdings möchten wir einige andere Lösungen kurz beleuchten, um diese mit dem PSO vergleichen zu können.
Da das Traveling Salesman Problem schon sehr lange existiert, gibt es einige Ansätze zur Lösung. Wir möchten uns aber speziell auf das *Brute-Force-Verfahren*, das *Nearest-Neighbor-Verfahren* und den *Ameisenalgorithmus* beschränken.

4.1 Problemstellung

Das Traveling Salesman Problem (TSP) ist eines der bekanntesten NP-vollständigen Probleme der Informatik. Dabei werden verschiedene Städte in einem vollständigen, gewichteten und ungerichteten Graphen simuliert.
Die Kanten entsprechen dabei den Straßenverbindungen zwischen den Städten und das Gewicht, je nach Definition, der Entfernung oder Fahrzeit zwischen diesen. Ziel ist es, den Hamiltonkreis mit dem niedrigsten Gesamtgewicht im Graphen zu finden.

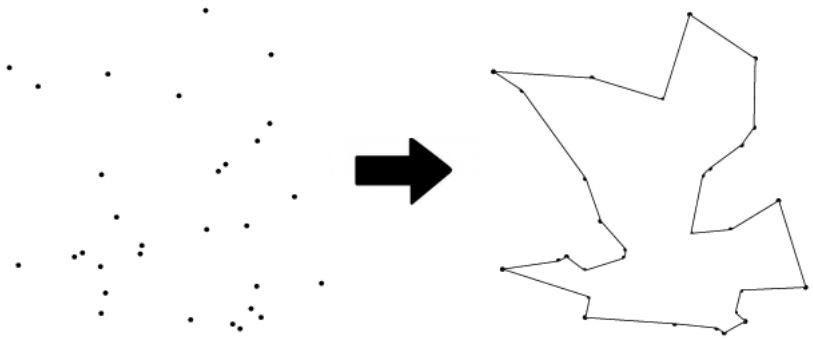

Abbildung 2: Das Traveling Salesman Problem [Wei16]

4.2 Das Brute-Force-Verfahren

Beim Brute-Force-Verfahren betrachtet man nacheinander alle möglichen Rundreisen, berechnet ihre Länge und sucht sich anschließend die kürzes-

te Reise heraus. Das Problem dieses Verfahrens ist, dass es bei steigender Anzahl an Städten sehr schnell ineffizient wird, obwohl es am Ende immer die kürzeste Route liefert [Hel16a].
Sei n die Anzahl der Städte. So haben wir $r(n) = (n-1)!\ /\ 2$ viele mögliche Rundreisen von unserem Startpunkt aus, da je zwei Reisen immer identisch sind und sich nur durch ihre Richtung unterscheiden.

n	$r(n)$
1	1
2	1
3	1
4	3
5	12
6	60
7	360
8	2520
9	20.160
10	181.440
11	1.814.400
12	19.958.400

Wie aus der oberen Tabelle ersichtlich ist, liegt beim Brute-Force-Verfahren eine Laufzeit vor, die schlechter als exponentiell (fakultativ) ist:

$$O((n-1)!\ /\ 2) = O(n!)$$

Für die spezielle Betrachtung der Laufzeit l gehen wir von einem Rechner aus, der $b = 1.000.000$ Berechnungen pro Sekunde erledigen kann. Dieser Rechner hätte für $n = 10$ Städte eine Laufzeit von

$$l = \frac{r(n)}{b} = \frac{181.440}{1.000.000} = 0,181s$$

Betrachtet man die Laufzeit für $n = 11$ Städte, so erkennt man, dass das Verfahren aufgrund der Fakultät eine $n-1$ mal lange Laufzeit hat, also

$$l = 0.181s \cdot 10 = 1,81s$$

Rechnet man nun dies nun auf 15 Städte hoch, so erhalten wir bereits eine Laufzeit von ca. $l = 726min = 12,1h$ [Hel16a].
Bedenkt man nun, dass in der Praxis oft nicht nur die zurückgelegte Strecke, sondern auch Reisekosten und Fristen relevant sind, ist das Brute-Force-Verfahren in sehr vielen Fällen nicht anwendbar.

4.3 Das Nearest-Neighbor-Verfahren

Das Nearest-Neighbor-Verfahren liefert nicht immer die optimale Lösung, hat aber im Gegensatz zum Brute-Force-Verfahren eine bessere Laufzeit von $O(n^2)$.
Es wird ausgehend vom Startknoten s immer die Kante mit dem geringsten Wert gewählt [Hel16b]. Mit dieser Methodik erhält man am Ende einen minimalen Spannbaum ausgehend von s. Dieser entspricht allerdings nicht immer der kürzesten Rundreise. Die Ergebnisse des Nearest-Neighbor-Verfahren werden aber trotzdem als für die meisten Anwendungen angemessen gute Ergebnisse erachtet, weshalb es in der Wirtschaft oft verwendet wird.

Beispiel 4.1 ***Nearest-Neighbor-Verfahren***

Abbildung 3: Rundreise: Erfurt-Dresden-Magdeburg-Potsdam

Ein Kaufmann plant eine Rundreise, bei der er die Städte Dresden, Magdeburg und Potsdam von Erfurt aus besuchen muss. Der nächste Nachbar von Erfurt ist Magdeburg. Somit ist die erste Kante, die ausgewählt wird, die Strecke zwischen Erfurt und Magdeburg. Führt man dies fort, erhält man folgenden minimalen Spannbaum:

Abbildung 4: MST der Rundreise

Somit erhalten wir als Ergebnis des Nearest-Neighbor-Verfahrens die Rundreise Erfurt-Magdeburg-Potsdam-Dresden-Erfurt mit einer Strecke von insgesamt 690km.

4.4 Der Ameisenalgorithmus

Der Ameisenalgorithmus (engl. Ant Colony Optimization) ist ein Optimierungsverfahren, das in den frühen 1990er Jahren nach dem Vorbild von Ameisenkollonien entwickelt wurde. Dabei wurde das Verhalten von Ameisen beobachtet, die von einem Punkt A aus mehrere mögliche Wege, um zum Punkt B zu gelangen, haben. Dabei fand man heraus, dass Ameisen, während sie eine Strecke zurücklegen, Pheromone ausstreuen und sich tendenziell für den Weg entscheiden, der die höchste Konzentration an Pheromonen besitzt.
Laufen also zu Beginn alle Ameisen vom Punkt A los, so wählen sie zufällig einen der Wege aus. Es wird jeder mögliche Weg von einer gewissen Anzahl von Ameisen begangen. Sind die Ameisen am Punkt B angelangt, so kehren sie auf ihrem Weg zum Punkt A zurück, verteilen dabei weiterhin Pheromone und erhöhen somit automatisch die Konzentration des Phermongehalts auf dem Weg.

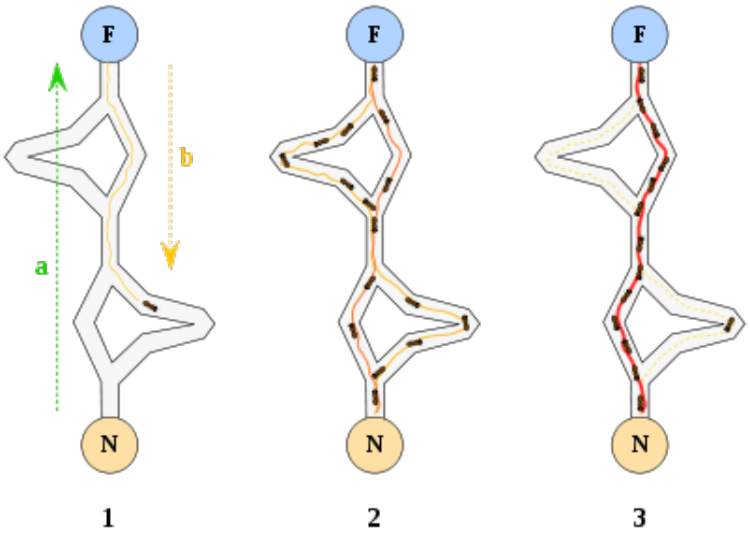

Abbildung 5: Der Ameisenalgorithmus [o.V16]

So ergibt sich automatisch, dass der schnellste Weg nach einer bestimmten Zeit die höchste Konzentration an Pheromonen besitzt, da die Ameisen, die diesen Weg gewählt haben, öfter als andere Ameisen die Strecke vom Punkt A zum Punkt B zurücklegen und sich immer mehr Ameisen an diesem Weg aufgrund des höheren Pheromonanteils orientieren [Blu05].
Mit einer Laufzeit von $O(z\ n^2\ m) = O(n^2)$ [Blu03] hat der Ameisenalgorithmus eine der besten Laufzeiten für die Lösung des Traveling Salesman Problems! Dabei stellt z die Anzahl der Zyklen, n die Anzahl der Städte und m die Anzahl der Ameisen dar. Somit erhalten wir selbst für 30 Städte mit der Wahl von 50 Zyklen und 300 Ameisen eine gute und schnelle Lösung:

$$O(z\ n^2\ m) = 50 \cdot 30^2 \cdot 300 = 13.500.000$$

Gehen wir nun von unserem zuvor angenommenen Rechner mit $b = 1.000.000$ Berechnungen pro Sekunde aus, so erhalten wir die Laufzeit

$$l = \frac{O(z\ n^2\ m)}{b} = \frac{13.500.00}{1.000.000} = 13,5s$$

Aufgrund dieser Schnelligkeit ist der Ameisenalgorithmus in vielen Unternehmen die bevorzugte Wahl zur Lösung von umfangreichen Optimierungsproblemen mit Graphen.

u

5 Anwendung der PSO am Traveling Salesman Problem

Da die Partikelschwarmoptimierung (PSO) auf beliebige Zielfunktionen angewandt werden kann, ist die Menge von Anwendungsmöglichkeiten sehr groß. So kann die PSO zum Beispiel auch zur Sortierung von beliebigen Mengen verwendet werden. Da für andere Problemstellungen oft keine effizienten Algorithmen bekannt sind, lässt sich die PSO bei diesen sinnvoll einsetzen. Ein Beispiel hiervon ist das Traveling Salesman Problem.

5.1 Definition

- **Städte C**
  Repräsentativ für die Städte, die es im Algorithmus zu verarbeiten gilt, werden Zahlen $c \in \mathbb{N}$ definiert.
  $C = \{1, ..., N\}$

  Wobei $N = |C|$ die Anzahl der zu verarbeitenden Städte darstellt.

- **Vollständiger Ungerichteter Graph G**
  Es wird ein Vollständiger Ungerichteter Graph G vorausgesetzt, dessen Knoten den Städten, dessen Kanten den Straßenverbindungen zwischen den Städten und dessen Gewichtungen den Längen der Straßen oder der Reisezeit zwischen den Städten entsprechen.

- **Position $p_i(t)$**
  Als Position $p_i(t)$ eines Partikels i zum Zeitpunkt t wird eine Permutation der Städte im Graphen definiert.

  $p_i(t) = (1, \ldots, N)$

  Die Reihenfolge der Städte entspricht der Reihenfolge, in der die Städte abgefahren werden können. Dementsprechend folgt nach der letzten Stadt N wieder die erste Stadt 1.

  Zum Startzeitpunkt wird für jedes Partikel x_i eine zufällige Reihenfolge für die Städte $p_i(0)$ festgelegt, die im Laufe des Algorithmus nach und nach verändert wird. Hierzu werden immer zwei Städte miteinander vertauscht.

Beispiel 5.1 *Ein Partikelschwarm bestehend aus 2 Partikeln sucht in einer Menge aus Städten* $S = \{1, ..., 7\}$. *Beide Partikel haben unterschiedliche, beliebig gewählte Startpositionen* $p_1(0) = (2, 1, 4, 5, 3, 7, 6)$ *und* $p_2(0) = (7, 1, 2, 5, 6, 3, 4)$.

- **Zielfunktion** Die Zielfunktion $f(p_i(t))$ wird definiert als die Summe aller Entfernungen zwischen je zwei aufeinanderfolgenden Städten, inklusive der Entfernung der letzten zur ersten Stadt.

  $$f(p_i(t)) = \sum_{u=1}^{n} \delta(u, \varphi_{i,t}(u))$$

  wobei u der u-ten Stadt in der Reihenfolge der Permutation von $p_i(t)$ und $\varphi_{i,t}(u)$ dessen Nachfolger in $p_i(t)$ entspricht. $\delta(u, \varphi_{i,t}(u))$ entspricht dem Gewicht der Kante von u nach $\varphi_{i,t}(u)$).

  Beispiel 5.2 *Folgendes Beispiel sei gegeben:*

  $C = \{1, 2, 3, 4\}$

  $1 \leftrightarrow 2 = 2$

  $1 \leftrightarrow 3 = 1$

  $1 \leftrightarrow 4 = 3$

  $2 \leftrightarrow 3 = 2$

  $2 \leftrightarrow 4 = 1$

  $3 \leftrightarrow 4 = 3$

  $p_i(t) = (3, 2, 4, 1)$

  Dann ist
  $f(p_i(t)) = 3 \leftrightarrow 2 + 2 \leftrightarrow 4 + 4 \leftrightarrow 1 + 1 \leftrightarrow 3 = 2 + 1 + 3 + 1 = 7$

- **Transposition a**
  Eine Transposition $a = \{i, j\}$ mit $1 \leq i, j \leq N; i \neq j$ entspricht einer Vertauschung der Reihenfolge zweier Städte in einer Position $p_i(t)$.

- **Geschwindigkeit** $v_i(t)$
  Die Geschwindigkeit $v_i(t)$ entspricht einer Liste von Transpositionen $a_i, ..., a_k$ an $p_i(t)$. In einer Geschwindigkeit können beliebig viele Transpositionen vorhanden sein, wobei hier die Reihenfolge von Bedeutung ist [Liu14]. Folgen zwei gleiche Transpositionen nacheinander, heben diese sich gegenseitig auf.

$v_i(t) = (a_i, ..., a_k)$ mit $1 \leq i, k \leq N$

Also können auch mehrere Städte innerhalb eines Zeitintervalls a vertauscht werden. Je "schneller" sich ein Partikel bewegt, desto mehr Städte werden in einer Iteration vertauscht.

Beispiel 5.3 *Es sei* $C = \{1, ..., 7\}$ *mit der Startposition des Partikels* i $p_i(0) = (1, 2, 3, 4, 5, 6, 7)$ *gegeben.*
Wird nun die Geschwindigkeit $v_i(t) = (\{1, 2\}, \{2, 3\})$ *auf die Position angewandt, erhält man die neue Position* $p_i(1) = (3, 1, 2, 4, 5, 6, 7)$*, da zuerst die Städte* 1 *und* 2 *und danach die Städte* 2 *und* 3 *vertauscht werden.*

- **Menge aller Transpositionen** Die Menge aller Transpositionen A_g wird definiert als die Menge aller möglichen Vertauschungen in p.
  $A_g = \bigcup_{k=1}^{N-1} \bigcup_{l=k+1}^{N} \{k, l\}$

5.2 Berechnungen

Für die Rechenoperationen können die Standardformeln aus Kapitel 3.2 von Seite 10 ohne Anpassung angewandt werden. Allerdings müssen die einzelnen Operationen auf das TSP angepasst werden.

5.2.1 Subtraktion

- **Position - Position = Geschwindigkeit**
  Position - Position beschreibt den Vektor, also die Geschwindigkeit, die benötigt wird um von einer Position zur anderen zu Gelangen. Sucht man also z.B. den Vektor $v_i(t) = p_1(t) \rightarrow p_2(t)$ von Position $p_1(t)$ zu Position $p_2(t)$ wird dies mit der Subtraktion $p_2(t) - p_1(t) = v_i(t)$ errechnet.

  Es gilt:
  $p_i(t) - p_k(t) = p_k(t) - p_i(t)$ (Assoziativ)
  $p_i(t) - p_k(t) = \emptyset$, wenn $p_i(t) = p_k(t)$.

  Die Geschwindigkeit von $p_1(t)$ nach $p_2(t)$ ist die Summe der Transpositionen von $p_1(t)$ und $p_2(t)$. Hierzu werden die einzelnen Stellen der Permutationen $p_1(t)$ und $p_2(t)$ als Transposition direkt auf $p_2(t)$ angewandt, sodass $p_2'(t)$ entsteht. Die nächste Stelle wird auf $p_1(t)$ und $p_2'(t)$

angewandt. Die Liste der in dem Algorithmus angewandten Transpositionen ergeben $v_i(t)$.

Beispiel 5.4 *Gegeben seien die Positionen $p_1(t)$ und $p_2(t)$ mit*

$$p_1(t) = (5, 7, 1, 3, 4, 6, 2)$$
$$p_2(t) = (7, 1, 5, 4, 2, 3, 6)$$

Gesucht sei $v_i(t) = p_1(t) \to p_2(t) = p_2(t) - p_1(t)$

Nun werden die Permutationen der Reihe nach verglichen und die Transpositionen auf $p_2(t)$ angewandt.

1 $p_1(t) = (5, 7, 1, 3, 4, 6, 2)$
$p_2(t) = (7, 1, 5, 4, 2, 3, 6) \to \{5, 7\}$

2 $p_1(t) = (5, 7, 1, 3, 4, 6, 2)$
$p_2'(t) = (5, 1, 7, 4, 2, 3, 6) \to \{1, 7\}$

3 $p_1(t) = (5, 7, 1, 3, 4, 6, 2)$
$p_2'(t) = (5, 7, 1, 4, 2, 3, 6) \to \{1, 1\} \to \emptyset$

4 $p_1(t) = (5, 7, 1, 3, 4, 6, 2)$
$p_2'(t) = (5, 7, 1, 4, 2, 3, 6) \to \{3, 4\}$

5 $p_1(t) = (5, 7, 1, 3, 4, 6, 2)$
$p_2'(t) = (5, 7, 1, 3, 2, 4, 6) \to \{2, 4\}$

6 $p_1(t) = (5, 7, 1, 3, 4, 6, 2)$
$p_2'(t) = (5, 7, 1, 3, 4, 2, 6) \to \{2, 6\}$

7 $p_1(t) = (5, 7, 1, 3, 4, 6, 2)$
$p_2'(t) = (5, 7, 1, 3, 4, 6, 2) = p_1(t)$

Ergibt $v_i(t) = (\{5, 7\}, \{1, 7\}, \{3, 4\}, \{2, 4\}, \{2, 6\})$

5.2.2 Multiplikation

- **Koeffizient · Geschwindigkeit = Geschwindigkeit**
  Wird eine Geschwindigkeit $v_i(t)$ mit einer Zahl c multipliziert, wird die Größe der Geschwindigkeit, also die Menge der Transpositionen, verändert. t_i sind in diesem Fall die einzelnen Transpositionen, die eine Geschwindigkeit $v_i(t)$ bilden. k ist die Anzahl der Transpositionen in $v_i(t)$.

  $cv_i(t)$, mit $v_i(t) = (a_1, a_2, ..., a_k); c \in \mathbb{R}$

– Wird V mit 0 multipliziert, wird die Resultierende Geschwindigkeit $v_i(t)$ keine Transpositionen mehr enthalten.

$c = 0 : cv = \emptyset$

Beispiel 5.5 $0 \cdot (\{1,2\},\{1,3\},\{4,5\},\{3,4\}) = \emptyset$

– Ist $0 < c \leq 1$, so wird die Menge der Transpositionen in $v_i(t)$ um den Faktor $1 - c$ verringert. Hierzu werden nur die Transpositionen t_1 bis $t_{\lfloor ck \rfloor}$ in die resultierende Geschwindigkeit übernommen. Also hat die resultierende Geschwindigkeit noch ck viele Transpositionen. Im Sonderfall $c = 1$ bleibt $v_i(t)$ unverändert, da $ck = c$ ist.

$c \in (0;1] : cv = t_1, ..., t_{\lfloor ck \rfloor}$

Beispiel 5.6 $0,5 \cdot (\{1,2\},\{1,3\},\{4,5\},\{3,4\}) = (\{1,2\},\{1,3\})$

– Bei einem $c > 1$ muss die Menge der Transpositionen vergrößert werden. Hierzu werden $k(c-1)$ Elemente aus V_g ausgewählt und mit $v_i(t)$ addiert.

$c > 1 : v_i(t) + \lfloor k(c-1) \rfloor \cdot$ Zufallsauswahl aus V_g

Beispiel 5.7 $1,5 \cdot (\{1,2\},\{1,3\},\{4,5\},\{3,4\})$
$= (\{1,2\},\{1,3\},\{4,5\},\{3,4\},\{1,5\},\{3,5\})$

– Bei einem negativen c wird $v_i(t)$ durch dessen Komplement $\bar{v}_i(t)$ ersetzt. $\bar{v}_i(t)$ ist hierbei die Menge V_g minus die Menge der Elemente der Liste $v_i(t)$. c selbst wird durch den positiven Betrag ersetzt.

$c < 0 : cv_i(t) = -c \cdot \bar{v}_i(t)$, mit $\bar{v}_i(t) = V_g \setminus v$

Beispiel 5.8 $S = \{1,2,3\}, V_g = \{\{1,2\},\{1,3\},\{2,3\}\}$
$-1,5 \cdot (\{1,3\}) = 1,5 * (\{1,2\},\{2,3\}) = (\{1,2\},\{1,3\},\{2,3\})$

5.2.3 Addition

- **Geschwindigkeit + Geschwindigkeit = Geschwindigkeit**
  Definition der Addition zweier Geschwindigkeiten:
  $v_1 = \{a_1^1, ..., a_1^n\}$
  $v_2 = \{a_2^1, ..., a_2^k\}$
  $v_1 + v_2 \leq \{a_1^1, ..., a_1^n, a_2^1, ..., a_2^k\}$

Zwei aufeinanderfolgende gleiche Transpositionen werden hierbei entfernt, da diese sich gegenseitig aufheben. Das heißt, wenn zwei Geschwindigkeiten addiert werden, wird die Liste der Transpositionen länger und das Partikel bewegt sich schneller. Beim Anwenden der resultierenden Geschwindigkeit auf eine Position $p_i(t)$ eines Partikels x_i zum Zeitpunkt t werden alle Vertauschungen der zwei ursprünglichen Geschwindigkeiten nacheinander angewandt.

Beispiel 5.9 *Wie im Beispiel auf Seite 20 werden wieder 7 Städte mit* $S = \{1, ..., 7\}$ *verwendet.*
Werden nun die Geschwindigkeiten $v_1(t) = (\{1, 2\}, \{2, 3\})$ *und* $v_2(t) = (\{5, 7\})$ *addiert, so erhält man die neue Geschwindigkeit* $v_{1+2}(t) = (\{1, 2\}, \{2, 3\}, \{5, 7\})$

- **Position + Geschwindigkeit = Position**
  Die Position $p_i(t+1)$ eines Partikels wird erreicht, indem alle Transpositionen in $v_i(t)$ an der Permutation $p_i(t)$ angewandt werden [Liu14].
  $p_i(t) = (1, ..., i, ..., j, ..., N)$
  $v_i(t) = (\{i, j\})$
  $p_i(t) + v_i(t) = (1, ..., j, ..., i, ..., N)$

  Dies entspricht der Anwendung der Geschwindigkeit auf eine Position, wie sie im Beispiel auf Seite 20 gezeigt wird.

5.3 Laufzeit

Mit dem Pseudocode für PSO auf Seite 12 dieser Arbeit lässt sich die Laufzeit des Algorithmus für die Lösung des TSP abschätzen.

Die Initialisierung des Algorithmus wird für jedes Partikel x_i durchgeführt und ist daher abhängig von der Anzahl der Partikel n. Das Initialisieren der Position ist von der Länge der Permutation und somit von N abhängig. Die Geschwindigkeiten sind aber von der Menge der möglichen Transpositionen abhängig, die im Bereich von N^2 wächst.

In der Schleife, die in Zeile Acht beginnt und die im Normalfall die größte Laufzeit des Algorithmus ausmacht, ist von der Anzahl der Iterationen abhängig. Da sich das nicht überprüfen lässt, wie gut das bisher beste gefundene Ergebnis des Algorithmus ist, lässt sich nicht voraussagen, wie viele Iterationen der Algorithmus durchlaufen muss. Die Anzahl der Iterationen ist daher ebenfalls nicht von der Anzahl der Städte abhängig. Zeile Neun

bis Dreizehn bilden wieder eine Schleife, deren Laufzeit von der Anzahl der Partikel abhängig ist. Die Zeilen, die entscheidend für die Laufzeit sind, sind die Zeilen Zehn und Elf, da beide von der Größe der Geschwindigkeiten und somit von der Größe V_g abhängig sind. V_g steigt quadratisch zu der Anzahl der Städte. Kürzt man nun die Faktoren, die für die Laufzeit irrelevant sind, erhält man ein Ergebnis von $O = (N^2)$, wobei N die Anzahl der Städte in G darstellt.

6 Weitere Anwendungsbeispiele

Die Partikelschwarmoptimierung kann abgesehen vom Traveling Salesman Problem noch auf eine Vielzahl weiterer Problemstellungen angewandt werden. Zwei davon möchten wir im Folgenden noch kurz anschneiden.

6.1 Motor

Ein Beispiel wurde zu Anfang der Arbeit bereits eingeführt: Die Optimierung der Motorleistung. Um das größte Drehmoment aus dem Motor herauszuholen, müssen einige Parameter angepasst werden. Bei einem modernen Benzinmotor mit Turbolader gibt es beispielsweise folgende Parameter:

1. Benzin-Luft Mischungsverhältnis m
2. Ladedruck p_l
3. Gaspedal / Drosselklappe g
4. Drehzahl d
5. Zündwinkel w
6. Ventilsteuerzeit t_v
7. Ventilverstellung v
8. Einspritzzeit t_e
9. Einspritzdruck p_e

Diese Parameter definieren den mehrdimensionalen Suchraum. Da die Parameter alle einen großen Wertebereich besitzen und sich teilweise gegenseitig beeinflussen ist es nur sehr schwer möglich einen Motor zu optimieren. Simulationen oder Tests auf einem Drehmoment Prüfstand liefern die Ergebnisse der Zielfunktion.

6.2 Ablaufplanung

Ablaufplanung findet in praktisch jedem Unternehmen statt. Dabei kann durch eine möglichst gute Planung eine Menge an Ressourcen gespart werden. Das Ziel dabei ist, dass man n Aufgaben möglichst fristgerecht oder zumindest mit einer möglichst geringen Verspätung abschließt. Wendet man die PSO auf diese Problemstellung an, so definieren wir folgende Werte:

Seien $j = 1, 2, \ldots, n$ die zu erledigenden Aufgaben. Dann besitzt jede Aufgabe j eine Bearbeitungszeit p_j, eine Frist d_j und eine Priorität w_j.

Der Ablauf der Aufgaben wird durch einen n-dimensionalen, geordneten Vektor $s = (s_1, s_2, \ldots, s_n)^T$ dargestellt, dem ein Prozessbeginn $j = 1, 2, \ldots, n$ zugeordnet wird. Dabei wird davon ausgegangen, dass zu Beginn alle Aufgaben zur Verfügung stehen.

Beispiel 6.1 *Eine Druckmaschine soll 3 verschiedene Magazine drucken. Sei nun* $s_1 = 2, s_2 = 1$ *und* $s_3 = 3$*. So wird das zweite Magazin als erstes, das erste Magazin als zweites und das dritte Magazin als letztes gedruckt.*

Möchte man nun die Fertigstellungszeit C_j einer Arbeit j wissen, so berechnet man diese über

$$C_j = \sum_{i \in \mathbb{N}, wobei s_i \leqq s_j} p_i$$

Die Fertigstellungszeit berechnet sich also durch die Summe der Bearbeitungszeiten der Aufgaben, die vor ihr bearbeitet wurden, inklusive der eigenen Bearbeitungszeit.
Demzufolge wird die Verspätung berechnet durch

$$T_j = max\{0, C_j - d_j\},$$

also durch die Differenz aus der Fertigstellungszeit der Arbeit und der Frist [PV10].

Möchte man seine Ablaufplanung mittels PSO optimieren, so durchsuchen wir den Suchraum nach der kürzesten Sequenz beziehungsweise nach der Sequenz, die die geringsten Verspätungen aufweisen.

7 Fazit

Betrachtet man die Partikelschwarmoptimierung, so lassen sich einige Vor- und Nachteile feststellen.
Zum einen ist sie durch ihre Blackbox-Funktion sehr anpassungsfähig und kann auf einem breitem Spektrum angewandt werden. Zum Beispiel wie gezeigt im Bereich der Ablaufplanung, oder auch in der Konstruktion von Motoren. Zum anderen ist sie relativ leicht zu implementieren, was in der praktischen Anwendung Zeit und auch Kosten spart.
Dadurch, dass die PSO zu den Approximationsverfahren gehört, sind die Ergebnisse, ebenso wie beim Nearest-Neighbor-Verfahren, nicht verifizierbar. Das heißt man kann nicht überprüfen, ob das optimale Ergebnis gefunden wurde oder nicht. Man muss also ausschließlich von Näherungslösungen mit einer bestimmten Qualität ausgehen, was bei speziellen Problemen dazu führt, dass die gelieferte Genauigkeit nicht genügt und somit die PSO keine Alternative zur Lösung dieser darstellt. Zudem gibt es für die Probleme, die man mit der PSO lösen kann, in den meisten Fällen bereits speziellere Algorithmen, die genauso leicht zu implementieren sind, keine Anpassung benötigen und eine bessere Laufzeit besitzen.

Alles in allem ist es aber durchaus möglich, das Traveling Salesman Problem per PSO effizient zu lösen. Es bietet sich allerdings an, den zuvor betrachteten Ameisenalgorithmus zu verwenden, da dieser speziell für diese Problemstellung entwickelt wurde und eine bessere Laufzeit besitzt. Dagegen ist die PSO für die Problemstellung eher geeignet, als das Brute-Force oder Nearest-Neighbor-Verfahren, da es schnellere und tendenziell bessere Ergebnisse liefert.

Vor allem aber sollte man die Partikelschwarmoptimierung ebenso wie alle anderen evolutionären Algorithmen zukünftig nicht aus den Augen verlieren, da diese aufgrund ihrer Flexibilität, ihrer guten Laufzeit und der immer weiter voranschreitenden Spezialisierung ein großes Potenzial zur Lösung zukünftiger Probleme aufzeigen.

Literatur

[Blu03] Daniel Blum. Ant colony optimization (aco). page 8, 2003. Accessed: 2016-05-21.

[Blu05] Christian Blum. Ant colony optimization: Introduction and recent trends. pages 354–356, 2005. Accessed: 2016-05-21.

[Cle12] Maurice Clerc. Standard particle swarm optimisation. *<hal-00764996>*, 2012.

[Ham12] Michael Hamann. Populationsbasierte metaheuristiken: Grundlagen und schwarmintelligenz. 2012.

[Hel16a] Ulrich Helmich. *TSP, Brut force-Methode*. 2016. Accessed: 2016-05-21.

[Hel16b] Ulrich Helmich. *TSP, Nearest neighbour-Methode*. 2016. Accessed: 2016-05-21.

[JK95] R. Eberhart J. Kennedy. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, page 1942–1948, 1995.

[Liu14] Yushan Liu. Partikelschwarmoptimierung fuer diskrete probleme. 2014. Accessed: 2016-05-23.

[Mel08] Jan Melchior. Implementierung von partikelschwarm-optimierung und vergleich mit einer evolutionsstrategie. 2008.

[o.V16] o.V. Ameisenalgorithmus. 2016. Accessed: 2016-02-21.

[PV10] Konstantinos E. Parsopoulos and Michael N. Vrahatis. Particle swarm optimization. *Particle Swarm Optimization and Intelligence: Advances and Applications*, pages 25–40, 186, 2010.

[Sch99] Walter Schmitting. *Das Traveling-Salesman-Problem*. 1999. Accessed: 2016-05-20.

[Sie15] Andreas Siegling. Entwurf, Implementierung und Analyse von Partikelschwarm-Algorithmen für das Sortierproblem. 2015.

[Wei16] Eric W. Weisstein. Traveling salesman problem. 2016. Accessed: 2016-05-21.